Laboratorio de Datos

Introducción a Python // Parte 03

2do. Cuatrimestre
2023

DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

exactas
UBA

Laboratorio de Datos

Introducción a Python - Parte 03

... por Manuela Cerdeiro (y modificaciones de P. Turjanski)

Contenido

- Numpy
- Pandas
- Ejercicios

Numpy

Numpy (Numerical Python)

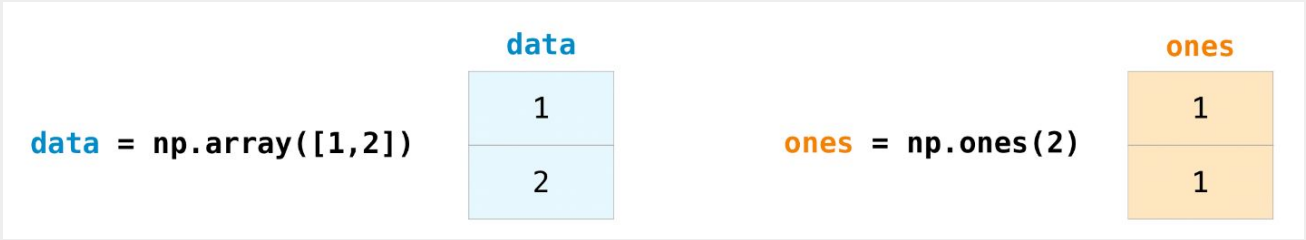
- ➔ Colección de módulos de código abierto que tiene aplicaciones en casi todos los campos de las ciencias y de la ingeniería.
- ➔ Estándar para trabajar con datos numéricos en Python.
- ➔ Muchas otras bibliotecas de Python (Pandas, SciPy, Matplotlib, scikit-learn, scikit-image, etc) usan numpy.
- ➔ Objetos: matrices multidimensionales por medio del tipo **ndarray** (un objeto n-dimensional homogéneo, es decir, con todas sus entradas del mismo tipo)
- ➔ Métodos para operar **eficientemente** sobre las mismas.

Se lo suele importar así:

```
import numpy as np
```

Numpy (Numerical Python)

```
import numpy as np

a = np.array([1, 2, 3, 4, 5, 6]) # 1 dimensión
b = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]) # 2 dimensiones
print(a[0])
print(b[0])
print(b[2][3])
print(b[2,3])
np.zeros(2) # matriz de ceros del tamaño indicado
np.zeros((2,3))

np.ones(2)
```

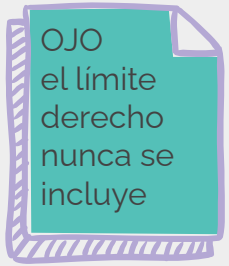


Numpy (Numerical Python)

También podés crear vectores a partir de un rango de valores:

```
np.arange(4) # array([0, 1, 2, 3])
```

También un vector que contiene elementos equiespaciados, especificando el primer número, el límite, y el paso.




```
np.arange(2, 9, 2) # array([2, 4, 6, 8])
```

También podés usar np.linspace() para crear un vector de valores equiespaciados especificando el primer número, el último número, y la cantidad de elementos:

```
np.linspace(0, 10, num=5) # array([0.,2.5,5.,7.5,10.])
```

Ejercicio

Generá un vector que tenga los números impares entre el 1 y el 19 inclusive usando arange().

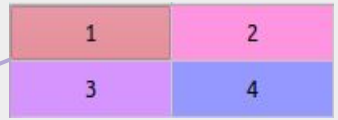
Repetí el ejercicio usando linspace(). ¿Qué diferencia hay en el resultado?

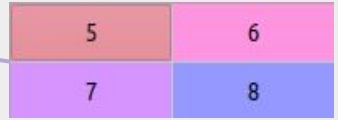
Ejemplos

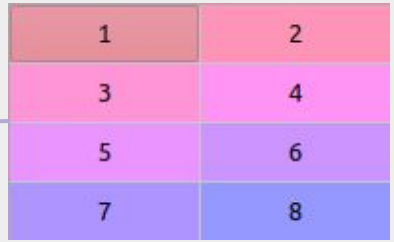
```
a = np.array([1, 2, 3, 4])
b = np.array([5, 6, 7, 8])
c = np.concatenate((a, b))
```

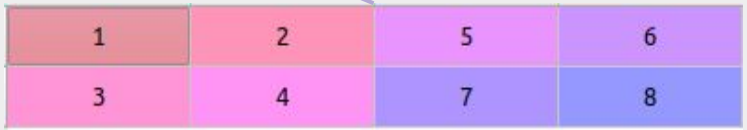
```
x = np.array([[1, 2], [3, 4]])
y = np.array([[5, 6], [7, 8]])
```

```
s = np.concatenate((x, y), axis = 0)
t = np.concatenate((x, y), axis = 1)
```







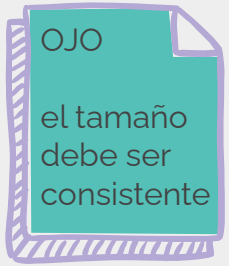


Ejemplos

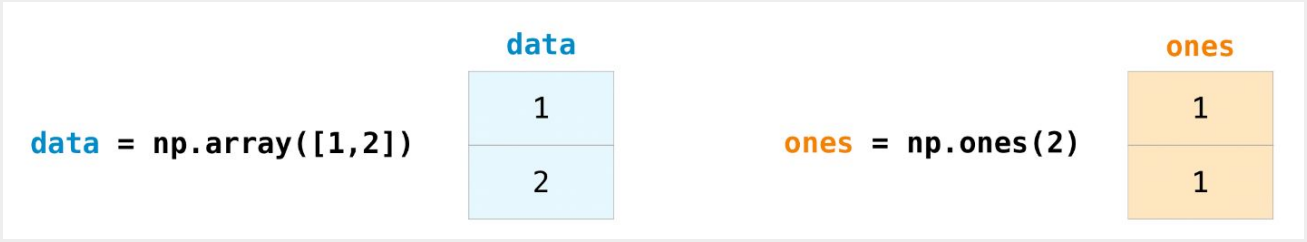
Un ejemplo de array de 3 dimensiones.

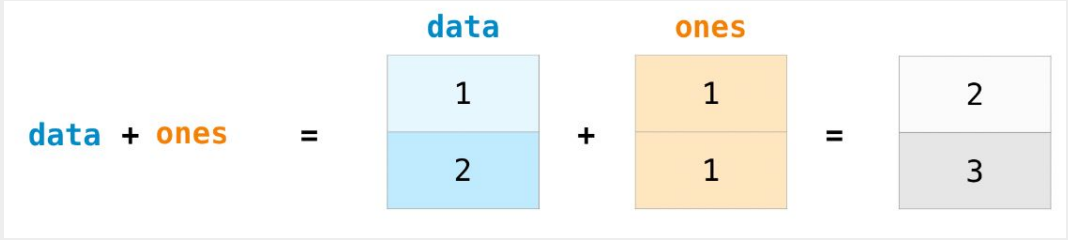
```
array_ejemplo = np.array([
                          [[0, 1,  2,  3],[4, 5, 6, 7]],
                          [[3, 8, 10, -1],[0, 1, 1, 0]],
                          [[3, 3,  3,  3],[5, 5, 5, 5]]
                         ])
```

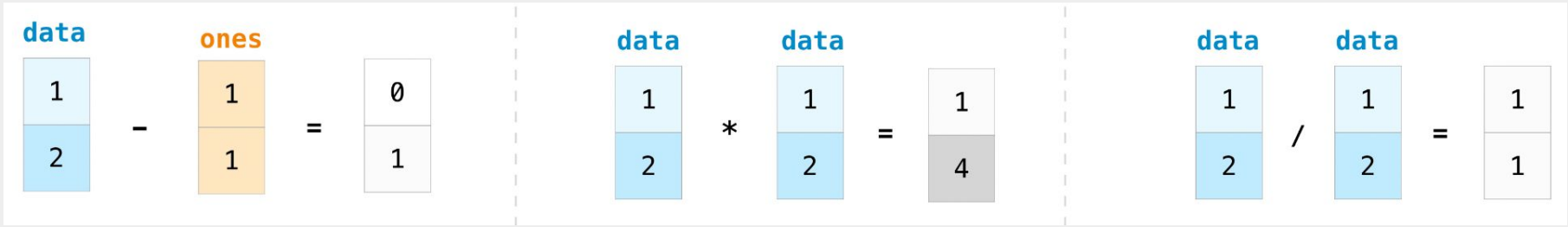
```
array_ejemplo.ndim # cantidad de dimensiones - 3
array_ejemplo.shape # cantidad de elementos en cada eje (3,2,4)
array_ejemplo.size # total de entradas 3*2*4

array_ejemplo.reshape((12,2)) # modifico la forma
array_ejemplo.reshape((4,6))
array_ejemplo.reshape((3,-1)) # 3 por lo que corresponda
```



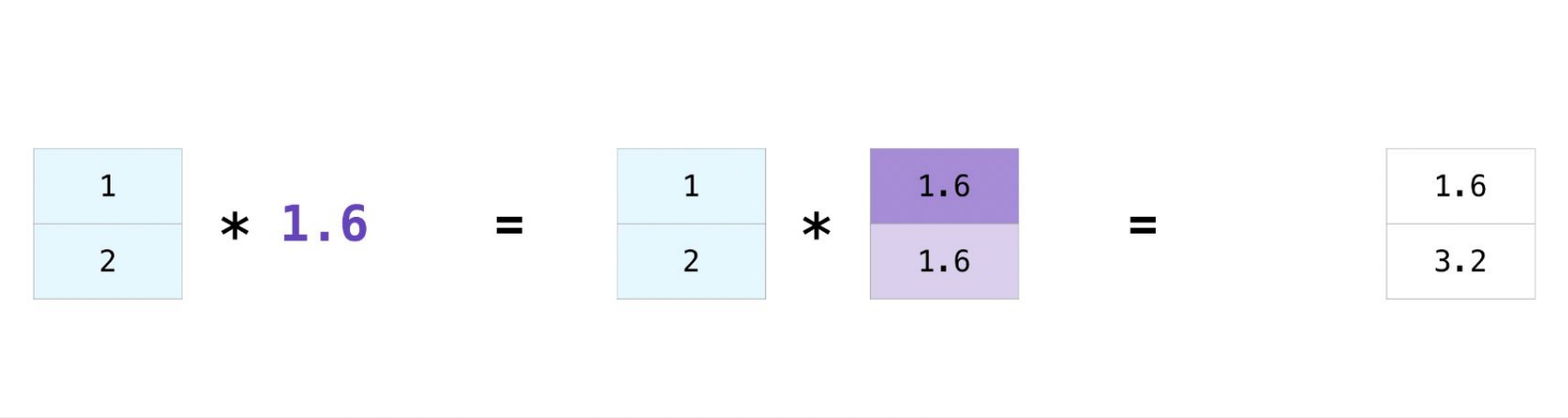
Operaciones

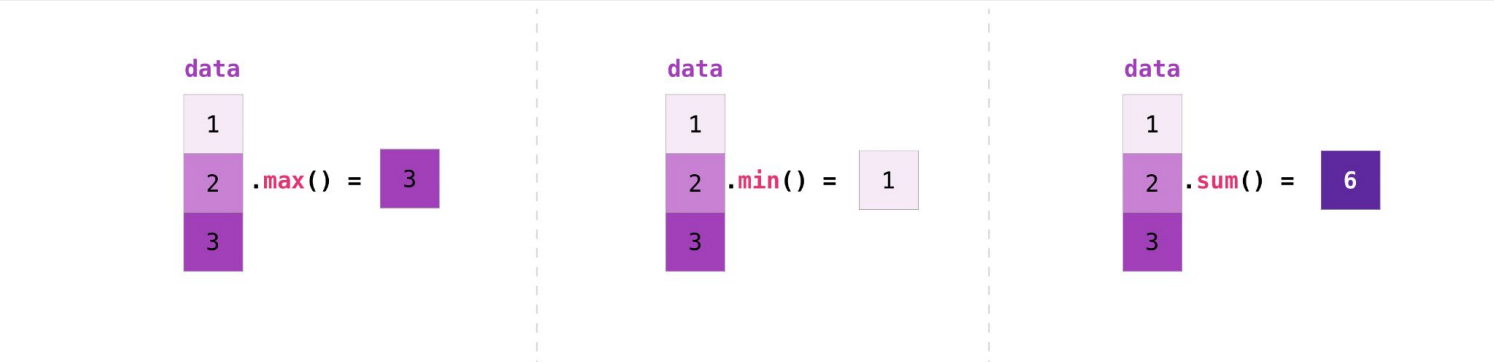




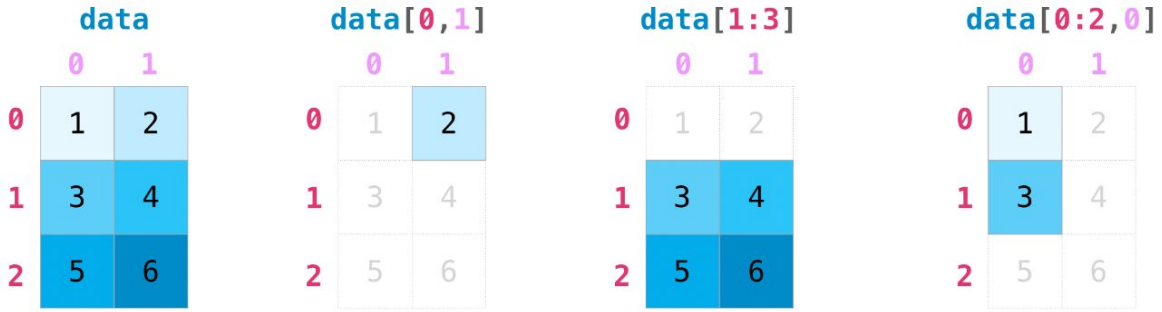


Operaciones

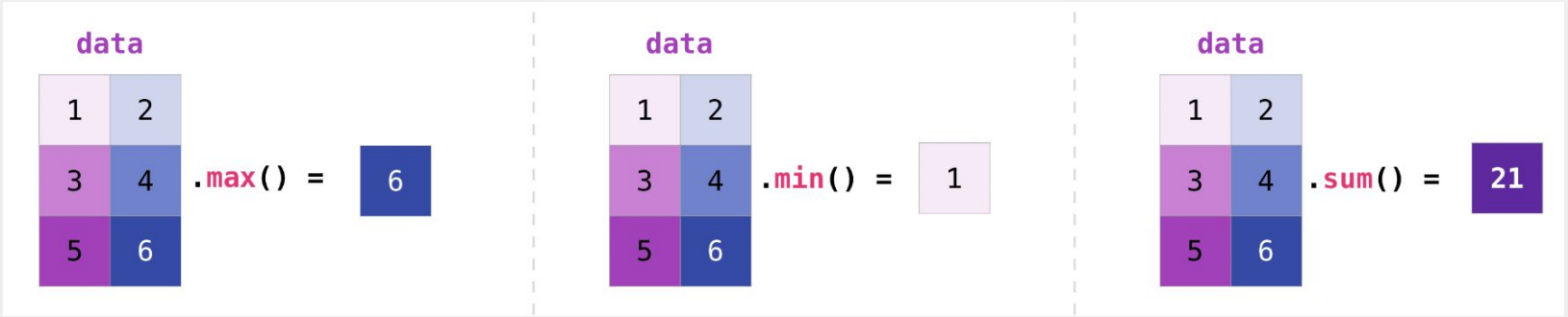


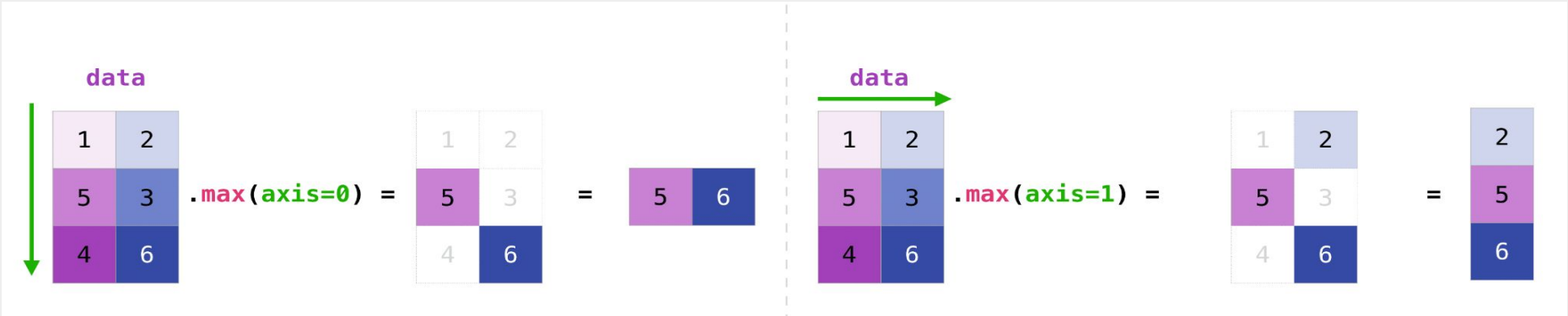


Operaciones



Operaciones





Ejercicios

Definir una función `pisar_elemento(M,e)` que tome una matriz de enteros `M` y un entero `e` y devuelva una matriz similar a `M` donde las entradas coincidentes con `e` fueron cambiadas por `-1`.

Por ejemplo si `M = np.array([[0, 1, 2, 3], [4, 5, 6, 7]])` y `e = 2`, entonces la función debe devolver la matriz `np.array([[0, 1, -1, 3], [4, 5, 6, 7]])`

Pandas

Pandas

- Pandas es una extensión de NumPy para manipulación y análisis de datos.
- Ofrece estructuras de datos y operaciones para manipular **tablas** de datos (numéricos y de otros tipos) y **series** temporales.
- Tipos de datos fundamentales: **DataFrames** que almacenan tablas de datos y las **Series** que contienen secuencias de datos.

```
import pandas as pd
```

Pandas

```
import pandas as pd

import os

archivo = 'arbolado-en-espacios-verdes.csv'

directorio = './Downloads/'


fname = os.path.join(directorio,archivo)

df = pd.read_csv(fname)
```

La variable `df` es de tipo `DataFrame` y contiene todos los datos del archivo csv estructurados adecuadamente.

Con `df.head()` podés ver las primeras líneas de datos. Si a `head` le pasás un número como argumento podés seleccionar cuántas líneas querés ver. Análogamente con `df.tail(n)` verás las últimas `n` líneas de datos.

Pandas

```
>>> df.head() # primeras líneas

   long       lat      id_arbol ...          origen       coord_x       coord_y

0 -58.477564 -34.645015    1  ...           Exótico  98692.305719  98253.300738

1 -58.477559 -34.645047    2  ...           Exótico  98692.751564  98249.733979

2 -58.477551 -34.645091    3  ...           Exótico  98693.494639  98244.829684

3 -58.478129 -34.644567    4  ...  Nativo/Autóctono  98640.439091  98302.938142

4 -58.478121 -34.644598    5  ...  Nativo/Autóctono  98641.182166  98299.519997
```

Pandas

```
>>> df.columns
Index(['long', 'lat', 'id_arbol', 'altura_tot', 'diametro', 'inclinacio',
       'id_especie', 'nombre_com', 'nombre_cie', 'tipo_folla', 'espacio_ve',
       'ubicacion', 'nombre_fam', 'nombre_gen', 'origen', 'coord_x',
       'coord_y'],
      dtype='object')
>>> df.index
RangeIndex(start=0, stop=51502, step=1)
```

Pandas

```
>>> df[['altura_tot', 'diametro', 'inclinacio']].describe()
         altura_tot      diametro    inclinacio
count  51502.000000  51502.000000  51502.000000
mean      12.167100     39.395616      3.472215
std        7.640309     31.171205      7.039495
min        0.000000      1.000000      0.000000
25%        6.000000     18.000000      0.000000
50%       11.000000     32.000000      0.000000
75%       18.000000     54.000000      5.000000
max       54.000000    500.000000     90.000000
```

Filtros

```
>>> df['nombre_com'] == 'Ombú'
0        False
1        False
2        False
3         True
...
>>> (df['nombre_com'] == 'Ombú').sum()
590
df['nombre_com'].unique()   # una vez cada nombre
```

Filtros

```
cant_ejemplares = df['nombre_com'].value_counts()

cant_ejemplares.head(10)  # tabla con los 10 nombres más frecuentes
```

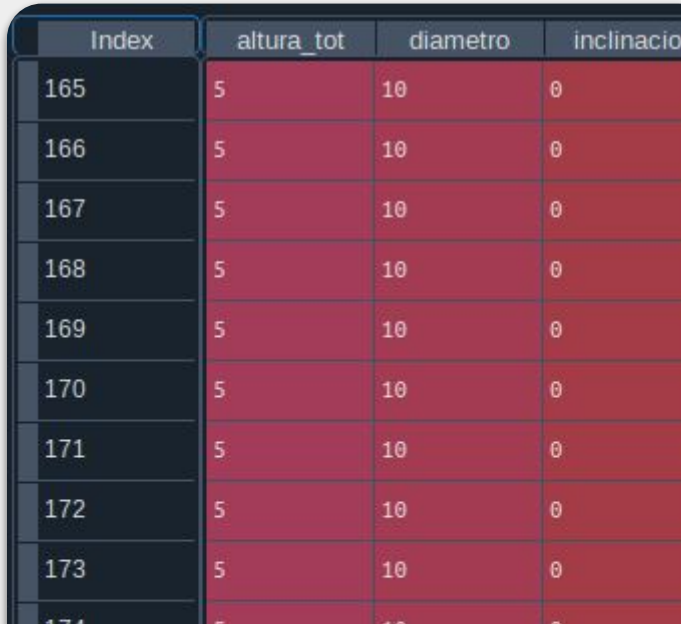
Filtros

```
>>> df_jacarandas = df[df['nombre_com'] == 'Jacarandá']

>>> cols = ['altura_tot', 'diametro', 'inclinacio']

>>> df_jacarandas = df_jacarandas[cols]

>>> df_jacarandas.tail()

       altura_tot  diametro  inclinacio

51104           7        97           4

51172           8        28           8

51180           2        30           0

51207           3        10           0

51375          17        40          20
```

Filtros + Copia

Si queremos modificar `df_jacarandas` es conveniente crear una copia de los datos de `df` en lugar de simplemente una vista. Esto se puede hacer con el método `copy()` como en el siguiente ejemplo.

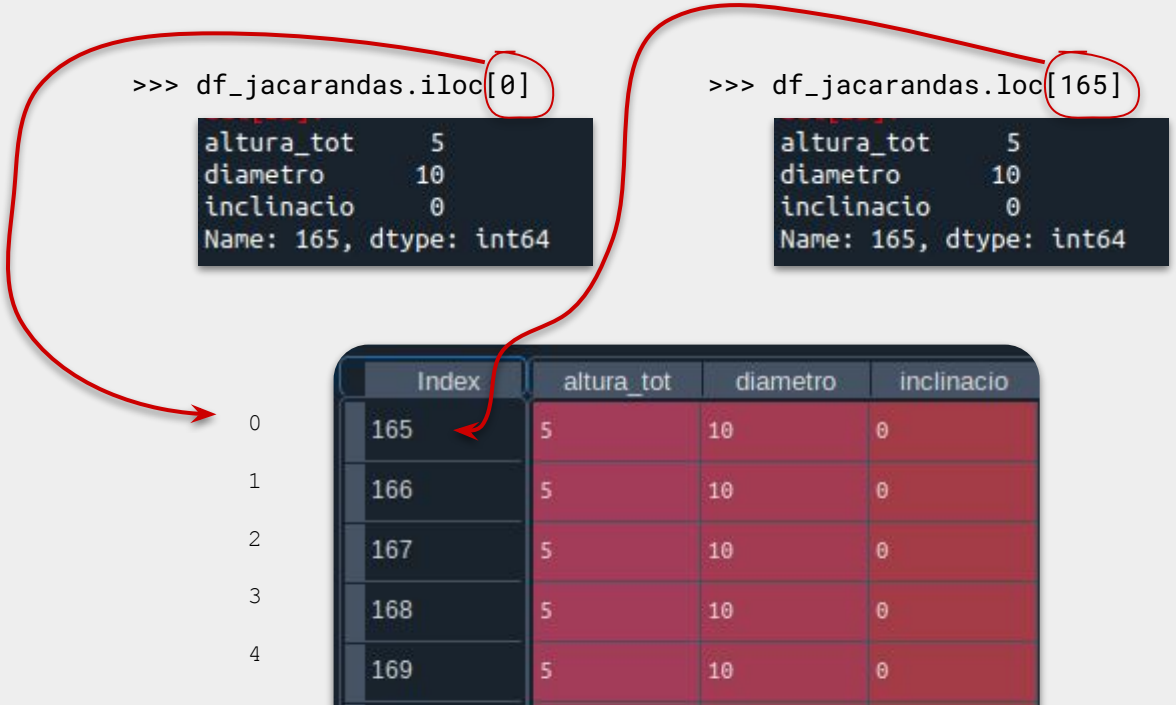
```
df_jacarandas = df[df['nombre_com'] == 'Jacarandá'][cols].copy()
```



Index	altura_tot	diametro	inclinacio
165	5	10	0
166	5	10	0
167	5	10	0
168	5	10	0
169	5	10	0
170	5	10	0
171	5	10	0
172	5	10	0
173	5	10	0

Filtros por posición // índice

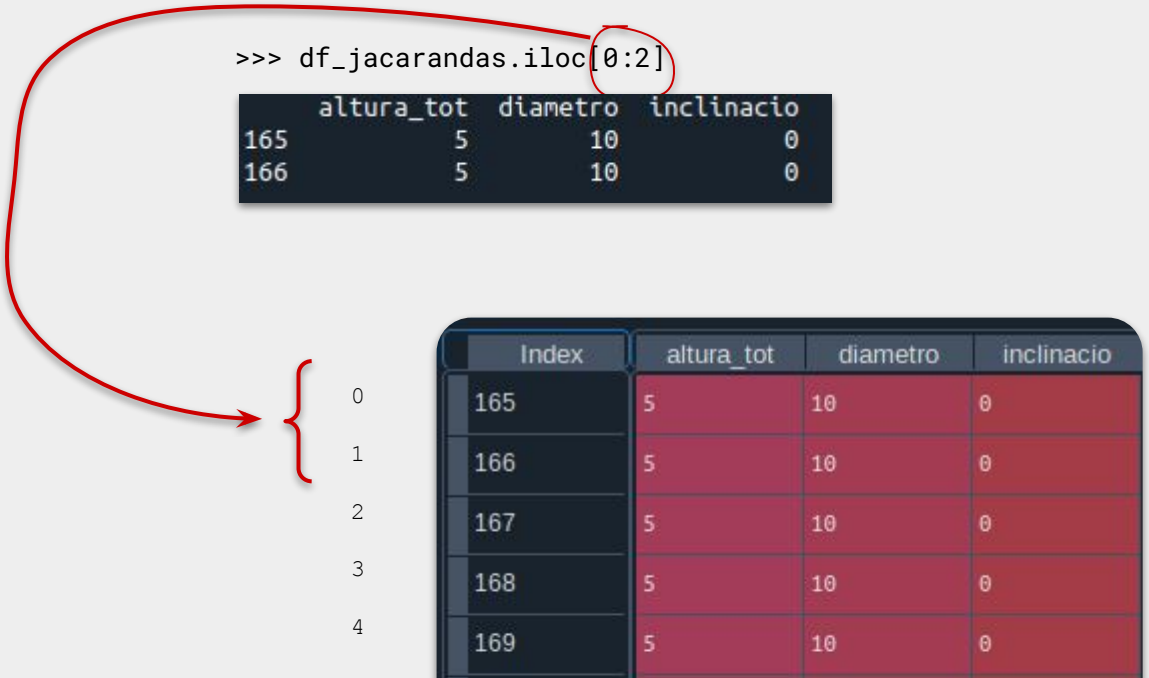
Al registro podemos accederlo tanto por *posición* con `iloc` (como hacíamos con los arrays) como por *índice* con `loc` (especie de clave)



Filtros por posición // índice

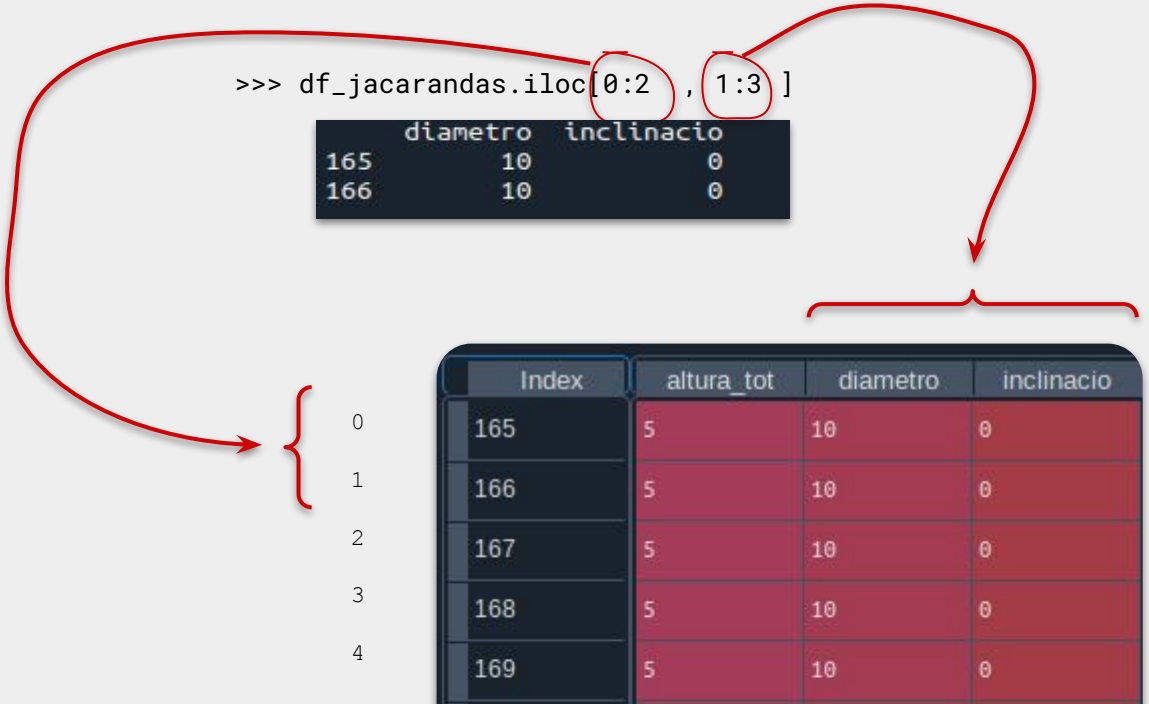
También podemos acceder a un slice

```
>>> df_jacarandas.iloc[0:2]
     altura_tot  diametro  inclinacio
165           5        10           0
166           5        10           0
```



Filtros por posición // índice

Podemos filtrar filas Y COLUMNAS



Filtros

Se puede seleccionar una sola columna especificando su nombre.
Importante. Al tomar una sola columna se obtiene una serie en lugar de un DataFrame:

```
>>> diametros = df_jacarandas['diametro']

>>> type(diametros)

pandas.core.frame.DataFrame

>>> type(diametros)

pandas.core.series.Series
```

Ejercicios

Ejercicios - Parte 1

A partir del dataset 'arbolado-en-espacios-verdes.csv' que contiene datos relacionados con el censo de arbolado realizado durante el año 2011 en la Ciudad de Bs. As. (también lo pueden descargar del campus), hacer lo siguiente:

1. Cargar la información del archivo csv en un dataframe denominado `data_arboles_parques`
2. Armar un dataframe que contenga las filas de los Jacarandás
3. Armar un dataframe que contenga las filas de los Palos Borrachos
4. Para cada uno de estos dos dataframes calcular:
    a. Cantidad de árboles;
    b. altura máxima, mínima y promedio;
    c. diámetro máximo, mínimo y promedio
5. Definir las siguientes funciones:
    a. `cantidad_arboles(parque)` que dado el nombre de un parque calcule la cantidad de árboles que tiene
    b. `cantidad_nativos(parque)` que dado el nombre de un parque calcule la cantidad de árboles nativos.

Ejercicios - Parte 2

A partir del dataset '[arbolado-publico-lineal-2017-2018.csv]', que contiene datos relacionados con el arbolado público, localizado en la traza de la Ciudad de Bs. As. (también lo pueden descargar del campus), hacer lo siguiente:

1. Cargar la información del archivo csv en un dataframe denominado `data_arboles_veredas`. El dataset debe tener **solamente** las siguiente columnas:

   ```
   cols_sel = ['nombre_cientifico', 'ancho_acera', 'diametro_altura_pecho', 'altura_arbol']
   ```

2. Imprimir las diez especies más frecuentes con sus respectivas cantidades
3. Trabajaremos con las siguientes especies seleccionadas:

   ```
   especies_seleccionadas = ['Tilia x moltkei', 'Jacaranda mimosifolia', 'Tipuana tipu']
   ```

   Una forma de seleccionarlas es la siguiente:

   ```
   df_lineal_seleccion = df_lineal[df_lineal['nombre_cientifico'].isin(especies_seleccionadas)
   ```

Ejercicios - Parte 3

Se quiere estudiar si hay diferencias entre los ejemplares de una misma especie según si los datos provienen de un dataset u otro (arbolado-en-espacios-verdes.csv vs. arbolado-publico-lineal-2017-2018.csv) . Para eso tendremos que relacionar datos de ambos dataframes.

El GCBA usa en el dataset arbolado-en-espacios-verdes.csv los nombres de columnas 'altura_tot', 'diametro' y 'nombre_cie' para las alturas, diámetros y nombres científicos de los ejemplares, y en el otro dataset (arbolado-publico-lineal-2017-2018.csv) usa 'altura_arbol', 'diametro_altura_pecho' y 'nombre_cientifico' para las mismas características.

También se puede observar que los nombres científicos varían entre ambos datasets. A modo de ejemplo 'Tipuana Tipu' se transforma en 'Tipuana tipu' y 'Jacarandá mimosifolia' en 'Jacaranda mimosifolia'. Obviamente son cambios menores pero suficientes para desalentar al usuarie desprevenide.

Ejercicios - Parte 3 (Cont.)

Proponemos los siguientes pasos para comparar los *diámetros a la altura del pecho* de las *tipas* en ambos tipos de entornos (datasets):

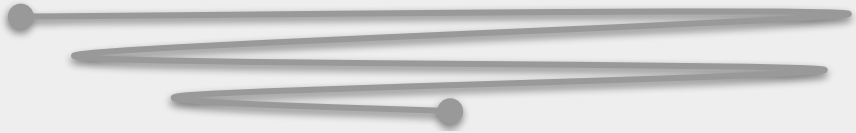
1. Para cada dataset armar uno nuevo (denominarlos `df_tipas_parques` y `df_tipas_veredas`, respectivamente) seleccionando sólo las filas correspondientes a las tipas y las columnas correspondientes al *diámetro*, a la *altura del pecho* y *alturas*.
   Importante. Hacerlo como copias (usando `.copy()` como aprendimos previamente) para poder trabajar en estos nuevos dataframes sin modificar los dataframes grandes originales.

2. Renombrar las columnas que muestran la altura y el diámetro a la altura del pecho para que se llamen igual en ambos dataframes.
   Ayuda. Explorar el comando `rename`.

3. Agregar a cada dataframe (`df_tipas_parques` y `df_tipas_veredas`) una columna llamada `'ambiente'`, que en un caso valga siempre `'parque'` y en el otro caso siempre `'vereda'`.

4. Juntar ambos datasets con el comando `df_tipas = pd.concat([df_tipas_veredas, df_tipas_parques])`. De esta forma tendremos en un mismo dataframe la información de las tipas distinguidas por ambiente.

Cierre



1. Numpy
2. Pandas
3. Ejercicios